EVENT HANDLING IN SPRING

You have seen in all the chapters that core of Spring is the **ApplicationContext**, which manages complete life cycle of the beans. The ApplicationContext publishes certain types of events when loading the beans. For example, a *ContextStartedEvent* is published when the context is started and *ContextStoppedEvent* is published when the context is stopped.

Event handling in the *ApplicationContext* is provided through the *ApplicationEvent* class and *ApplicationListener* interface. So if a bean implements the *ApplicationListener*, then every time an *ApplicationEvent* gets published to the ApplicationContext, that bean is notified.

Spring provides the following standard events:

S.N.	Spring Built-in Events & Description
1	**ContextRefreshedEvent** This event is published when the *ApplicationContext* is either initialized or refreshed. This can also be raised using the refresh() method on the *ConfigurableApplicationContext* interface.
2	**ContextStartedEvent** This event is published when the *ApplicationContext* is started using the start() method on the *ConfigurableApplicationContext* interface. You can poll your database or you can re/start any stopped application after receiving this event.
3	**ContextStoppedEvent** This event is published when the *ApplicationContext* is stopped using the stop() method on the *ConfigurableApplicationContext* interface. You can do required housekeep work after receiving this event.
4	**ContextClosedEvent** This event is published when the *ApplicationContext* is closed using the close() method on the *ConfigurableApplicationContext* interface. A closed context reaches its end of life; it cannot be refreshed or restarted.
5	**RequestHandledEvent** This is a web-specific event telling all beans that an HTTP request has been serviced.

Spring's event handling is single-threaded so if an event is published, until and unless all the receivers get the message, the processes are blocked and the flow will not continue. Hence, care should be taken when designing your application if event handling is to be used.

Listening to Context Events:

To listen a context event, a bean should implement the *ApplicationListener* interface which has just one method **onApplicationEvent()**. So let us write an example to see how the events propagates and how you can put your code to do required task based on certain events.

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Create Java classes *HelloWorld*, *CStartEventHandler*, *CStopEventHandler* and *MainApp* under the *com.tutorialspoint* package.
4	Create Beans configuration file *Beans.xml* under the **src** folder.
5	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of **HelloWorld.java** file:

```
package com.tutorialspoint;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

Following is the content of the **CStartEventHandler.java** file:

```
package com.tutorialspoint;

import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStartedEvent;

public class CStartEventHandler
   implements ApplicationListener<ContextStartedEvent>{

   public void onApplicationEvent(ContextStartedEvent event) {
      System.out.println("ContextStartedEvent Received");
   }
}
```

Following is the content of the **CStopEventHandler.java** file:

```
package com.tutorialspoint;

import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStoppedEvent;

public class CStopEventHandler
   implements ApplicationListener<ContextStoppedEvent>{

   public void onApplicationEvent(ContextStoppedEvent event) {
      System.out.println("ContextStoppedEvent Received");
   }
}
```

Following is the content of the **MainApp.java** file:

```
package com.tutorialspoint;

import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ConfigurableApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");

      // Let us raise a start event.
      context.start();

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();

      // Let us raise a stop event.
      context.stop();
   }
}
```

Following is the configuration file **Beans.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean  >
       <property name="message" value="Hello World!"/>
   </bean>

   <bean
         />

   <bean
         />

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
ContextStartedEvent Received
Your Message : Hello World!
ContextStoppedEvent Received
```

If you like, you can publish your own custom events and later you can capture the same to take any action against those custom events. If you are interested in writing your own custom events, you can check Custom Events in Spring